Wang Tiles for Image and Texture Generation

Michael F. Cohen[1], Jonathan Shade[2], Stefan Hiller[3], Oliver Deussen[3]
Microsoft Research[1], Wild Tangent[2], Technische Universität Dresden[3]

Abstract

We present a simple stochastic system for non-periodically tiling the plane with a small set of Wang Tiles. Wang tiles are squares in which each edge is assigned a color. A valid tiling requires all shared edges between tiles to have matching colors. We use this construction to create 2D textures containing discrete coherent objects that overlap tile edges. These types of patterns do not perform well with other texture synthesis procedures. We demonstrate a system for interactively designing the 2D tiles. We also apply automatic texture synthesis procedures to create 2D textures for the tiles. In another variation, a modification of Poisson disc sampling, 2D sampling patterns are created that can then be used to generate an infinite arrangement of plants on a terrain. We also show how the individual tiles containing plants can be preprocessed into a set of multiresolution view-dependent layered depth images for subsequent interactive rendering.

CR Categories and Subject Descriptions: I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques

Additional Keywords: non-periodic tiling, texture synthesis, point-based representations, level of detail algortihms

1 Introduction

Modeling and rendering scenes that capture the complexity of the real world is a difficult and time consuming task. A now well-known means to overcome this is to create (or capture) a small example of complexity and then reuse this example many times. Unfortunately, when the same example is used many times, the repetition is often apparent and distracting. We show how a small set of Wang tiles [28, 29] with either 2D or 3D textures plus a stochastic process for assembling them can be used to overcome this problem. Wang tiles are a set of squares in which each edge of each tile is "colored". Matching colored edges are aligned to tile the plane.

Recent work in texture synthesis exemplifies a means to avoid the repetition problem in two dimensions by using a small texture tile as an example to then create a larger amount of non-repetitive texture that has the same visual properties [6, 19, 31]. Unfortunately, these methods begin to break down as the size of coherent and recognizable objects within the texture increases. We show how a small set of 2D textures applied to Wang tiles avoids this. Stam was the first to consider the use of Wang tiles for texture synthesis [27]. We extend his work by presenting a simple interactive tool for designing the Wang tiles. We also demonstrate automated methods for defining the tiles as a set of textured primitives synthesized from a source image. We also create infinite Poisson-like distributions by placing points in the Wang tiles. This allows us to assemble infinite fields of objects such as in the sunflower field above.

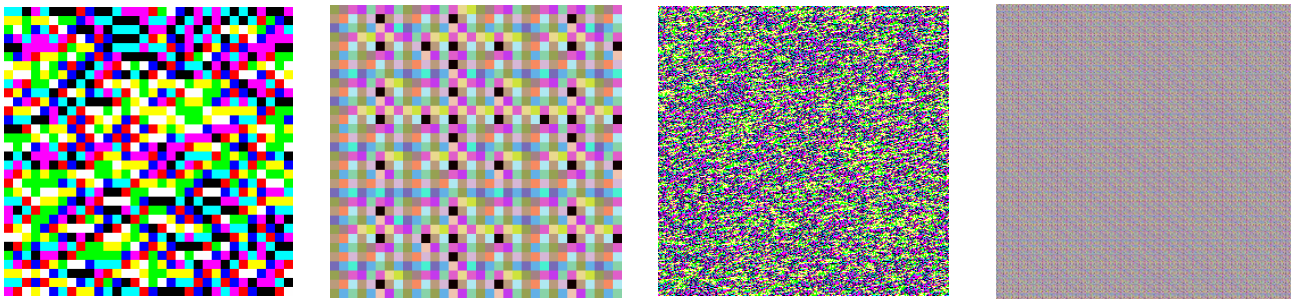
Street Address and Electronic Mail Address

Figure 1: A Wang tiled field filled with sunflowers.

Besides creating 2D textures, Wang tiles can also be applied to the field of image based rendering (IBR) [7, 14, 26]. IBR methods reuse pixels from one or more images to create new images. A related set of methods use point based primitives that may be sampled from three dimensional models[15, 21, 24]. Whether the points are derived by sampling a three dimensional model or by the inherent sampling process in image creation, these methods are highly dependent on the sampling and reconstruction processes. We will discuss how to create 3D sampled representations from 3D models anchored in 2D Wang tiles. These can then be used to interactively render highly complex scenes while avoiding most repetition artifacts. We show a system based on multiresolution view-dependent layered depth images that leverages both the original LDI image based rendering method and the Wang tile construction.

2 Wang Tiles

Wang tiles consist of a set of square tiles with color-coded edges. The squares cannot be rotated. A *valid* tiling of the infinite plane consists of any number of copies from the set laid down such that all contiguous edges have matching colors. This tiling gained its name based on a conjecture made in 1961 by Hao Wang. He conjectured that any set of tiles that can produce a valid tiling of the plane can also produce a *periodic* tiling the plane [8]. Five years later Berger [1] refuted Wang's conjecture that no *aperiodic* (can *only* tile aperiodically) set exists by constructing such a set containing 20426 tiles. He soon after developed an aperiodic set with 104 tiles, and until recently, the smallest known aperiodic set had 16 prototiles. The current record is a set containing 13 tiles discovered by Karel Culik [2, 12]. Wang tiles encompass many interesting features to theoreticians. The discovery of aperiodic sets of Wang tiles laid the groundwork for proving that it was



(a) (b) (c) (d)

Figure 2: Assemblies of pseudocolored tiles comparing stochastic tilings with strict aperiodic sets, (a) 32x32 stochastic (b) 32x32 Robinson aperiodic set of 16 tiles (c) 256x256 stochastic (d) 256x256 Robinson.

undecidable whether a given set of tiles can tile the plane. Wang tiles have recently been used to simulate the properties of proposed DNA computers. They have even taken a leading role in a science fiction story, "Wang's Carpets" by Greg Egan, that describes a life form based on Wang tile-like growth. Perhaps one of the most interesting aspects of Wang tiles that has prompted the uses just described is that sets of Wang tiles can mimic the behavior of any Turing machine.

We will not draw on any of these interesting aspects of Wang tiles, but rather on their simplicity, that they are square and that a small set can lead to a *non-periodic* tiling of the plane. We use the term *non-periodic* to indicate a tiling which is not periodic (although the set itself could lead to a periodic tiling), as opposed to the term *aperiodic* that refers to a set of tiles that cannot produce a periodic tiling. We also introduce a new stochastic process to laying down individual tiles. While the small aperiodic sets discussed in the theoretical literature typically have a deterministic assembly procedure, we purposefully introduce redundancy into the tile set construction. This provides the ability to have a stochastic process guide the actual layout of the tiles. This also results in tilings that have a more random appearance as can be seen in Figure **2**.

Figure 3: Eight Wang tiles that can stochastically tile the plane

We will work with sets that can be trivially shown to be able to tile the plane and trivially shown to not be aperiodic. In fact, a single tile, such as tile b in Figure 3 can tile the plane on its own. When tiling the plane, each choice of tile typically has two constraints. If one is placing tiles from West to East and from North to South[1] then each tile placed must have N and W edges that match the E and S edges already placed. If there are K colors then there are K^2 combinations of colors for two adjacent edges of the tiles (e.g., the south (S) and east (E) edges). So long as there is at least one tile in the set with each north (N) and west (W) combination, then the following simple procedure produces a valid tiling of any portion of the plane.

1. begin with any tile for the NW corner
2. tile the top row left to right by choosing tiles for which the W edge of the new tile matches the E edge of the previous tile
3. begin the next row with a tile such that the N edge matches the S edge from above
4. continue this row by randomly selecting tiles for which the N and W edges match the S and E edges from above and the left respectively (since our set contains all NW combinations, this is always possible)
5. go to step 3 for as many rows as desired

If the set contains at least *two* tiles with each NW combination, then there are always at least two choices at each step. If this choice is made with uniform probability, then the plane will always be tiled *non-periodically* since each step reduces to an independent random process (i.e., a coin flip). One such set of 8 tiles with 2 colors is shown in Figure 3. (Note that there are, in practical terms, only two colors since the red and green used for the NS edges could be replaced with the blue and yellow of the EW edges with no change in the set.) Using three colors results in the need for 18 tiles ($2 * 3^2$), however this can reduce repetition artifacts considerably as in Figure 7b.

In Figure 3 tiles a through d contain all NW combinations as do tiles e through h. The SE edge combinations are also each represented twice thus tilings can proceed from south to north as well. Many other sets are possible with all required properties. A valid tiling of a small portion of the plane may look like this.

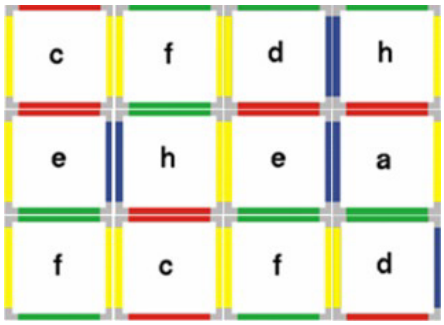


Figure 4: A small portion of the plane with a valid tiling

3 Tile Design

Wang tiles provide the framework for constructing large expanses of texture from a small amount of input. We now explore multiple ways for constructing the tiles themselves. We will first demonstrate and discuss an interactive tile design system. We

[1] We use the cardinal directions, N, S, E, W, to label edge directions.

follow this with an automated system that takes discrete texture primitives, such as images of leaves, as input. Because of the interdependence of tiles when primitives overlap tile boundaries we introduce an optimization method to evenly distribute primitives.

3.1 Interactive Tile Design

The simplest way to design a set of Wang tiles is to create them interactively. We have created a tool that allows one to place geometric primitives within the set. The primitives can be imported from drawing tool as images as long as the background is set to *transparent*. When a primitive is being placed in a tile, if it overlaps an edge, then the primitive automatically appears in all tiles with the same and opposing edges. Thus primitives can reside partially in one tile and partially in tiles with the opposite edge of the same color. Figure 5 shows a snapshot of the tool, the current set of tiles and a portion of the infinite plane. Figure 6 shows another set and resulting texture created interactively.

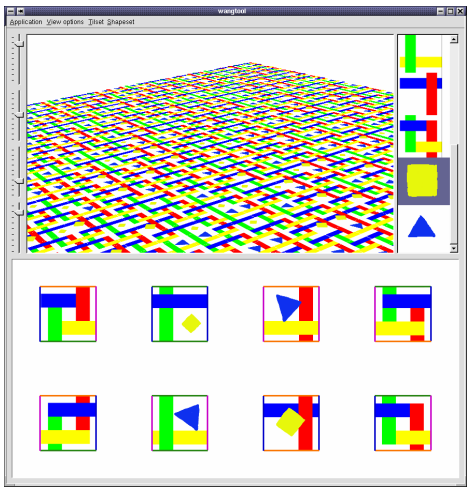


Figure 5: Interactive system for designing tiles.

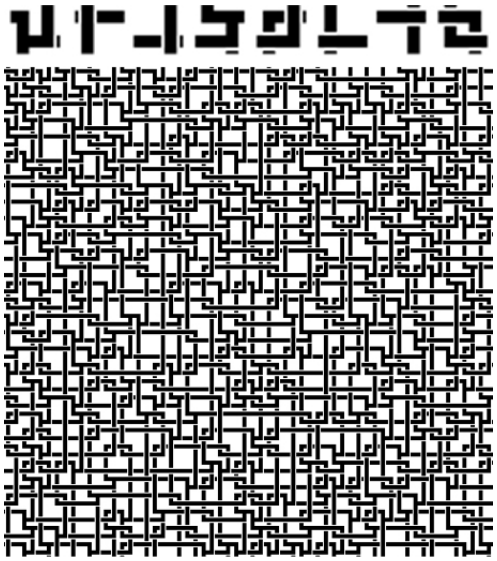


Figure 6: Interactively designed 8 tile set

3.2 Automatic Tile Design

The set of Wang tiles can also be generated automatically. We describe two methods. The first is a *texture synthesis* method derived from recent work by Efros and Freeman [6]. The second method relies on a Poisson disc-like distribution to place a set of large non-intersecting primitives (in this case sunflowers).

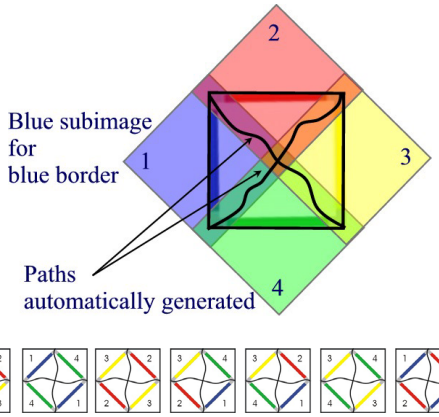


Figure 7: Four subimages are combined to form each Wang tile (tiles are rotated 45° in bottom row).

Texture Synthesis with Wang Tiles

To create a continuous texture from a sample, textures must be found for each tile that fit together across boundaries with matching colors. We use a variant of the algorithm proposed by Efros to generate such textures. To texture each tile, *2K* diamond shaped (squares rotated by 45 degrees) sample portions of the source image are combined, one for each edge color N-S and E-W (e.g., four for the eight tiles in Figure 3). In the example above we wish to texture a tile with one edge of each color. The four corresponding sample diamonds are assembled with an overlap as in Figure 7. The borders of the tiles are formed by the diagonals in the samples. For each tile a set of four diagonal paths is found that combines the pairs of diamonds optimally. This is done using the optimization algorithm given in [6]. During optimization, the errors of all paths within all tiles are minimized, thus paths are generated between all possible adjacent colors.

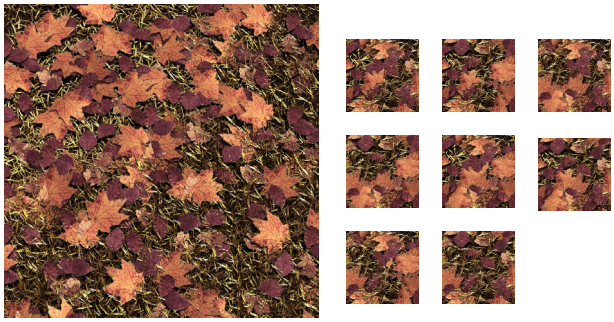


Figure 8: Source image and eight Wang tiles automatically generated based on set in Figure 3.

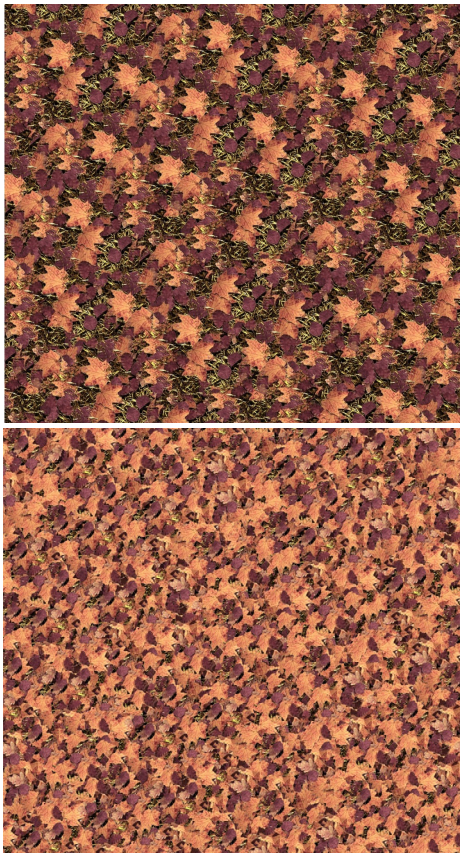


Figure 9: Small portion (64 tiles) of infinite textures from (a) set of eight automatically generated Wang tiles above, and (b) eighteen tile set utilizing 3 colors for the edges (from a different source image).

If the north edge of a tile is colored red, the lower diagonal part of the "red" diamond is used for synthesizing the upper part of tile, and vice versa for a red south edge. If this method is applied to all tiles and all colors, corresponding portions of the same diamond are guaranteed to lie across matching edges and the tiles fit together automatically.

This still leaves the interior portion of the tile free to have additional portions of the source added independently to each tile to privide further individualization.

Tile sets from Poisson distributions

For complex ecosystems like those in [3] tens of thousands of plants have to be distributed. Instancing can minimize the amount of geometry needed but even the storage of plant instances can consume lots of memory for large plant populations. An alternative is to position a few plants on a small set of Wang tiles and then generate the full population by combining the tiles non-periodically. Biological and environmental forces do not lead to random placements of plants but rather in a Poisson disc distribution to avoid crowding. Such positions can be obtained using a variant of Lloyd's method [4, 9 16] that was already published in Hiller et al [10] as an approach to sampling. A random point distribution is spread over each tile. The Voronoi diagram is calculated and the points are moved towards the center of gravity of their voronoi areas (Lloyd's method).

Using Wang tiles, the challenge is to find positions on each tile that generate a Poisson disc distribution for the whole population. The points near the border of a tile have influences across many tiles. If we assume the plants being placed have some radius, then any point within a radius of the border must also be included in all other tiles with a matching colored edge on the same side. The same point must also be considered to lie just outside any tile with an opposite matching colored edge. For this reason, standard Poisson dart throwing will not work (i.e., throw a dart at the tiles and let it stick if it's disc does not overlap others, even across tile boundaries). To obtain tiles that fit together we use a variant of Lloyd's method. Points are moved towards a position that is the average of all Voronoi areas that arise from this point in all tiles. One caveat is that we want to avoid any points whose radius overlaps a corner as this would cause the need for that point to exist in all tiles. Details and a numerical evaluation of this optimization process can be found in Hiller et al [10] or Deussen [4]. Figure 10 shows the results of this process on eight Wang tiles. Figures 1, 12 and 13 show this process applied to simulating a sunflower field.

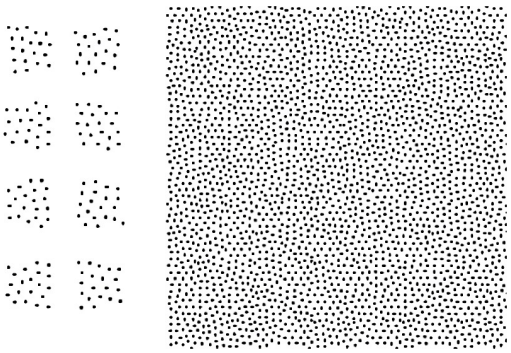


Figure 10: Eight Wang tiles with joint Poisson distributions, a portion of the plane resulting from these tiles.

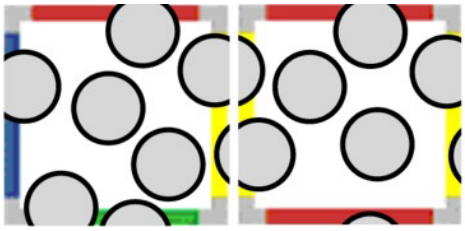


Figure 11: Two possible tiles with Poisson discs.

4 Three Dimensional Applications

The manual effort required to model natural environments and the computational cost required to render such scenes are both very high. To date, systems that attempt to render realistic looking natural environments have solved either one problem, or the other, but not both. The systems presented in Weber and Penn [30], Mech and Prusinkiewicz [18], Deussen et al. [3], and Prusinkiewicz et al. [22] are capable of creating realistic plants and environments. However, the models created by these systems are of such high detail that they can not be rendered in real-time. At the other end of the spectrum, geometric level-of-detail algorithms [11, 20] have increased the effective geometric complexity that can be rendered in real-time, but provide no direct support for realistic shading.

We can get the best of both worlds by using a level-of-detail

technique in combination with pre-calculated shading. However, pre-calculating the shading for each model individually leads to an explosion in data. In our particular case, a field of sunflowers, we would need to have a separate lighting condition calculated for every situation each of the instanced flowers appears in. To overcome this problem, we use tiles to group a set of flowers into a single lighting environment. We simply need to precalculate the lighting for each flower in each of the 8 tiles – a much smaller problem.

Point-sample-based representations are a natural solution for this problem [21, 24, 26]. In particular, Layered Depth Images (LDIs) allow us to convert complex geometry into a sparse set of samples with pre-calculated shading that can be rendered in real-time. We extend LDIs by introducing a novel view-dependent multi-resolution formulation of the representation that provides control of level-of-detail.

The system we have built relies on realistic models of plants and flowers have already been created. In particular, we use eleven variations of a model of a sunflower. The complete system consists of four steps: creating tiles composed of instances of the eleven sunflower models, computing a tiling of the plane that is then draped over a heightfield-based terrain, creating LDIs of each of the tiles, and finally, rendering the terrain and the tiles that make up the tiling.

4.1 Creating the Tiles

The flowers on each tile are positioned using the Poisson disc method outlined in the previous section. Sunflowers that intersect a tile boundary are repeated in all tiles with opposite matching colors. Figure 12 shows two of the eight tiles used to render all of the sunflower fields shown.

Figure 12: Two Wang tiles filled with sunflower geometry.

4.2 Draping a Tiling Over a Heightfield

To put the sunflowers in a realistic setting, we drape a tiling of sunflowers over a heightfield-based terrain. The tiling is created (using the stochastic algorithm outlined in Section 2) such that the corners of each tile in the tiling map to a single quadrilateral section of the heightfield. At render time, the heightfield is sampled to perturb the tiling so that it conforms to the terrain.

To create a benchmark for our real-time rendering system, we created a short raytraced video (accompanying the paper), a frame of which is seen in Figure 1. The image in Figure 1 differs from the LDI results in two ways. First, we are not able to render the trees in real time, so they are omitted. Second, the heightfield in the real-time system is sampled only at the corners of a tile and a shear warp applied to each half of a tile (split along a diagonal).

4.3 Creating LDIs of Wang Tiles

An LDI is essentially a sparse three dimensional volumetric data set in which two dimensions of the volume are parameterized by the image plane of a camera, while the thrid dimension is free to contain as many samples as are needed to represent the object being sampled. One of the primary advantage of LDIs is that the McMillan occlusion-compatible warp ordering [17] can be used to render the volume of samples in real-time, in software, without the use of a z-buffer.

The standard LDI defined by Shade et al. [26] is a perspective volume defined by a pinhole camera where each sample in the volume contains a color, a normal, and a depth. Since the objects we are sampling are rectangular volumes we will use an orthographic variant of the LDI in which the image plane of the camera is mapped to the top of a tile, and the free dimension lies along the vertical axis of the tile (and the flowers).

Multi-resolution View-dependent LDIs

The reconstruction quality of an LDI is typically dependent on the proximity of the novel viewpoint to the defining viewpoint of the LDI. For our orthographic volume, we wish to allow the viewpoint to allow views of the LDI from a complete ring of directions. We have two choices: populate it with enough samples that it looks correct from any viewpoint, or create a set of volumes that each look correct from a different viewpoint and choose which one to display based on the viewer's position. The first option creates volumes that are so dense they can't be rendered in real time: when viewing the front of a flower, why bother rendering the samples that make up the back of the flower? Our system instead constructs eight separate LDIs of each tile sampled from eight evenly spaced directions encircling the tile.

In addition, since we are creating expansive environments, the same tile may be seen from close-up and far-away in the same frame. Our system, thus also creates instances of each tile at seven levels of detail (resolutions of the volume). This is done for each of the direction-dependent LDIs resulting in 56 LDIs created for each original Wang tile. With eight tiles in a set, there are 448 LDIs created. Resolutions range from 700x700 for the highest to 65x65 for the lowest resolution.

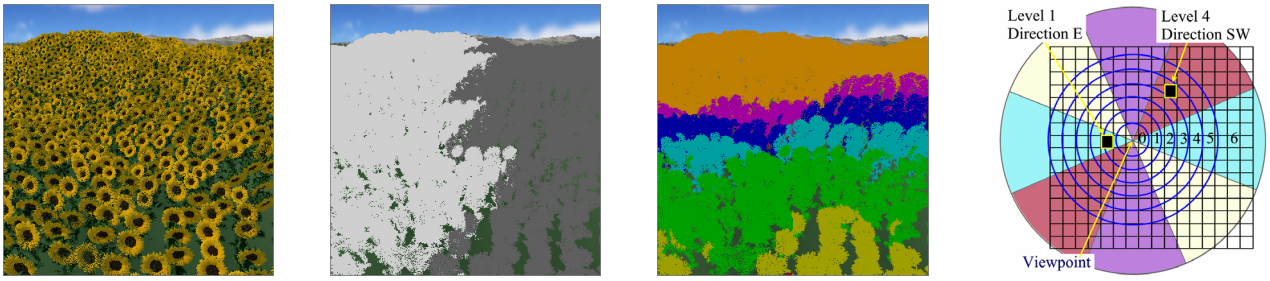


Figure 14: LOD selection. (a) A typical portion of the sunflower field. (b) This view straddles two of the 8 wedges that divide the space of directions around a tile. (c) Levels of detail false-colored. Red (not seen below us) is highest detail, orange is lowest detail. (d) Layout of directions and resolution levels from viewpoint.

LDI Sampling Strategy

The real-time renderer allows a virtual viewer to roam around a terrain covered in sunflowers. We optimize for a viewer to be a fixed "head" height above the terrain (a small distance above the sunflowers). We also fix the output filed of view and resolution.

The closest the virtual viewer can get to a tile is standing directly over the center of the tile, looking down upon it. Ideally, each sample in this volume would project to a single pixel in the output image. The seven levels of detail refer to tiles which are at distances 0 through 6 tile widths (more distant tiles use the lowest resolution). The resolution of each level of detail is chosen by placing the output perspective camera at head height the appropriate distance away and projecting the top of the tile onto the output image and measure its projection size in pixels. Figure 14 (b) shows a rendering of the sunflower field in which each the sunflowers have been false colored according to the level of detail chosen for a tile.

We are now ready to sample the volume of each tile into each LDI level and direction. We do this using a modified version of the public domain raytracer Rayshade [13]. Ideally, we would like to only store samples that will be visible at run-time. To this end, we choose a set of rays to cast towards each tile that mimics the rays of a typical virtual viewer. We also embed each tile in eight other tiles to simulate visibility.

We create an set of sampling rays on a cylinder to mimic the possible view directions. As the image plane of a virtual observer orbits a tile it is at all times tangential to a circle. We extend the circle above and below the height of the observer into a short cylinder. This will generate rays that simulate the viewer viewing the tile from greater and lesser heights that result from an uneven terrain. To ensure that we uniformly cover the space of rays between the cylinder and the tile volume, we employ a stratification of the cylinder (in two dimensions) and the tile volume (in three dimensions). Rays are cast from each stratum of the cylinder towards each stratum of the volume.

In summary, to create the LDIs for each tile we do the following. For each level of detail we place the tile at the origin and construct a cylinder, centered at "head height", with the appropriate radius, that encircles the tile. For each of eight directions we clear the volume of samples, divide one eighth of the cylinder into MxN strata and divide the volume into XxYxZ strata. For each stratum of the cylinder we cast a ray towards each stratum of the volume. The start and end positions of the ray are chosen with uniform distribution over the strata. Each time a ray intersects geometry in the tile, a sample is recorded in the appropriate LDI.

For comparison, we created three versions of the sunflower field: one with 6 flowers per tile, one with 12 flowers per tile, and another with 20 flowers per tile (the same density as in the raytraced image in Figure 12). All three data sets were sampled using the same parameters, the only difference among them is the density of the geometry in the tiles. In each case M,N,X, and Y were set to 18. Z, the dimension corresponding to height, was set to 6 for the closest tile. We employ a heuristic optimization to simulate the fact that nearby tiles will occlude almost all but the tops of distant tiles. We reduce the number of Z strata by one for each level of detail, with a minimum of 2 strata. Under this scheme, each level one LDI was sampled with 629,856 rays, each level two LDI was sampled with 524,880 rays, and so on. The sampling for each data set took approximately 30 hours on a workstation with an Intel Pentium III running at 1 GHz and 512 MB of memory. The data sets are 192MB, 266MB, and 326 MB in size.

4.4 Rendering the Terrain

The real-time renderer uses a combination of software rendering for the LDIs and hardware rendering for the texture-mapped terrain and sky. Rendering proceeds in three stages: first, the LDIs are rendered; second, the terrain is rendered and the resulting z-buffer read from the hardware; third, the hardware z-buffer is used to modify the image produced by the LDI renderer which is then alpha-composited into the frame-buffer of the graphics card.

We limit the field that is drawn to a 30x30 section of the tiling surrounding the viewer with new distant tiles added as the viewer moves. The tiling is rendered back-to-front. Each tile is shear-warped based on the terrain and projected into an RGBA image using the method of Shade et al. In addition to the standard LDI rendering, we also create a write-only z-buffer as we render the LDIs. Since the LDIs are rendered front-to-back always writing into the z-buffer (never reading and comparing) is correct. We will use this z-buffer to combine the result of LDI rendering with hardware rendering of the terrain.

Using OpenGL, we then render the sky and terrain, then read the z-buffer from the graphics card. Using this z-buffer, we iterate over every pixel in the image produced by LDI rendering. At any pixel where the terrain z-buffer is closer than the LDI z-buffer, we set the LDI image to have an alpha of zero (which prevents that pixel from being composited into the final image). Lastly, we alpha-composite the LDI image onto the hardware framebuffer and display the framebuffer. There are many different ways to combine software rendering and hardware rendering. We chose this method because we found that reading the z-buffer was much faster than writing the z-buffer.

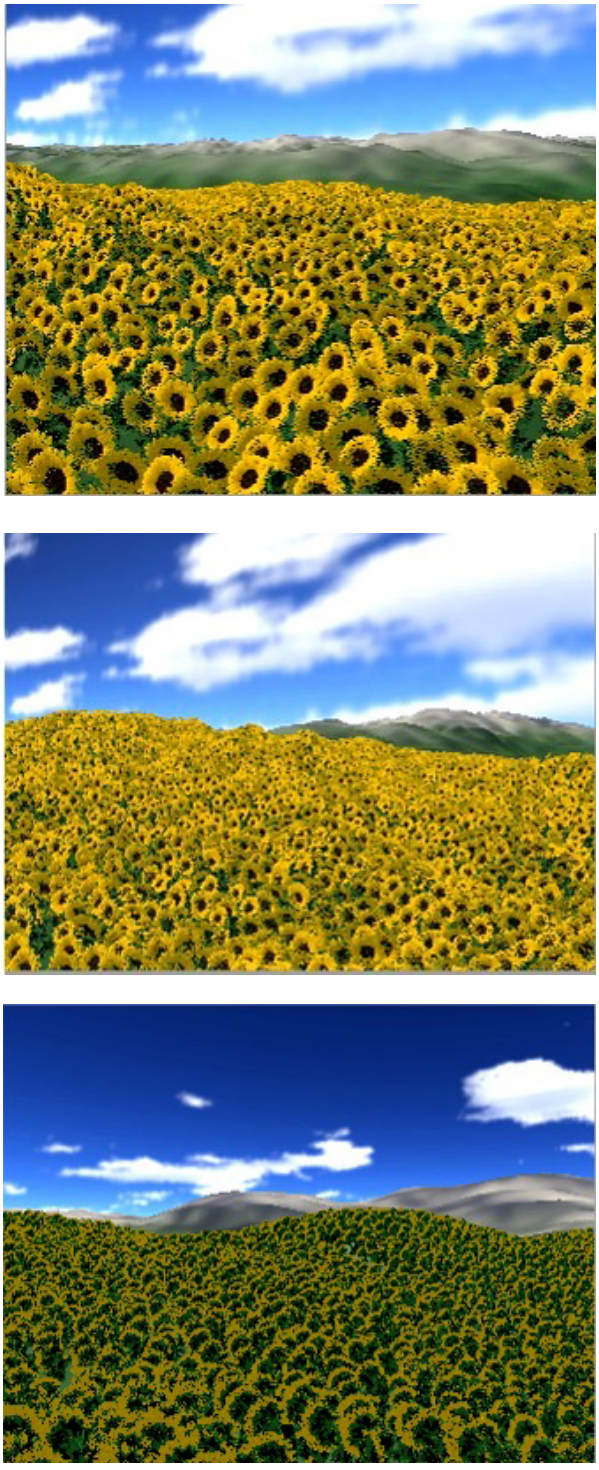


Figure 15: Frames from interactive LDI renderer with 20 flowers per tile. Sky is environment map. Mountains are OpenGL geometry. Frames rendered at between 2.7 and 4.0 fps on a P4 1.7 GHz PC.

We are able to render the sunflower field in real-time on a PC. Figure 14 gives rendering rates for two representative modern PCs: a 1.33 GHz AMD Athlon with an Nvidia GeForce2 MX graphics card and a 1.7 GHz Intel Pentium 4 with an Nvidia GeForce3 graphics card. Across the data sets, our software renderer was able to maintain a rendering rate of between 4.5 and 5.7 million depth pixels per second. Given the emerging point-rendering capabilities of modern graphics hardware, namely point-sprites in Microsoft's DirectX 8 graphics API, we believe point-sampled tiles can reach 30 fps.

5 Discussion

We hope we have shown the versatility and usefulness of Wang tiles for image and texture synthesis. Wang tiles were interesting in their own right as a theoretical entity. They should also now prove their usefulness in a variety of graphics applications.

We look forward to seeing how well a hardware point based renderer works with the LDI tiles. We also plan to extend the simple tiles containing only one type of primitive to scenes with many varieties of plants. We are certain there are many other applications of these elegant entities we have not thought of.

References

[1] R. Berger. The Undecidability of the Domino Problem. In *Memoirs Amer. Math. Soc*., 72, 1966.

[2] K. Culik II, An Aperiodic Set of 13 Wang Tiles. *Discrete Mathematics*, 160:245-251, 1996.

[3] O. Deussen, P. Hanrahan, B. Lintermann, R. Mech, M. Pharr, and P. Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. In *Proceedings of SIGGRAPH 1998*, 275-286.

[4] O. Deussen, S. Hiller, K. van Overveld, and T. Strothotte, Floating Points: A Method for Computing Stipple Drawings, In *Proceedings of Eurographics 2000 Conference.*

[5] T. Duff. Compositing 3-d Rendered Images. In *Proceedings of SIGGRAPH 1985,* 41-44.

[6] A. Efros and W. Freeman. Image Quilting for Texture Synthesis. In *Proceedings of SIGGRAPH 2001*, 341-346.

[7] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *Proceedings of SIGGRAPH 1996*, 43-54.

[8] B. Grunbaum and G. C. Shepard. *Tilings and Patterns.* W. H. Freeman and Company, 1987.

[9] A. Hausner. Simulating Decorative Mosaics. In *Proceedings of SIGGRAPH 2001*, 573-580.

[10] S. Hiller, O. Deussen, and A. Keller, Tiled blue noise samples, in *Proceedings of Vision Modelling Visualization 2001.*

[11] H. Hoppe. Progressive Meshes. In *Proceedings of SIGGRAPH 1996*, 99-108.

[12] J. Kari. A Small Aperiodic Set of Wang Tiles. *Discrete Mathematics*, 160:259-264, 1996.

[13] C. Kolb. *Rayshade User's Guide and Manual*. http://graphics.stanford.edu/~cek/rayshade, 1992.

[14] M. Levoy and P. Hanrahan. Lightfield Rendering. In *Proceedings of SIGGRAPH 1996*, 31-42.

[15] M. Levoy and T. Whitted. The Use of Points as Display Primitives. Technical Report TR 85-022, University of North Carolina, 1985.

[16] M. McCool and E. Fiume, Hierarchical Poisson Disk Sampling Distributions, *Proceedings of Graphics Interface, 1992.*

[17] L. McMillan. Computing Visibility Without Depth. Technical Report 95-047, University of North Carolina, 1995.

[18] R. Mech and P Prusinkiewicz, Visual Modeling of Plants Interacting With Their Environment. In *Proceedings of SIGGRAPH 1996*, 397-410.

[19] F. Neyret and M-P. Cani, Pattern-based Texturing Revisited. In *Proceedings of SIGGRAPH 1999,* 235-242.

[20] F. Perbet and M-P. Cani, Animating Prairies in Real-Time. In *Symposium on Interactive 3D Graphics 2001*.

[21] H. Pfister, M Zwicker, J van Baar, and M Gross. Surfels: Surface Elements as Rendering Primitives. In *Proceedings of SIGGRAPH 2000*, 335-342.

[22] P. Prusinkiewicz, L Mundermann, R. Karwowski, and B. Lane. The Use of Positional Information in the Modeling of Plants. In *Proceedings of SIGGRAPH 2001*, 289-300.

[23] R. M. Robinson Undecidable Tiling Problems in the Hyperbolic Plane. In *Inventiones Math*, 259-264, 1978.

[24] S. Rusinkiewicz and M. Levoy. Qsplat: A Multiresolution Point Rendering System. In *Proceedings of SIGGRAPH 2000*, 343-352.

[25] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture Mapping Progressive Meshes. In *Proceedings of SIGGRAPH 2001*, 409-416.

[26] J. Shade, S. J. Gortler, L. He, and R. Szeliski. Layered Depth Images. In *Proceedings of SIGGRAPH 1998*, 231-242.

[27] J. Stam. Aperiodic Texture Mapping. Technical Report R046. European Research Consortium for Informatics and Mathematics (ERCIM), 1997.

[28] H. Wang. Proving Theorems by Pattern Recognition. *Bell Systems Tech. J.*, 40:1-42, 1961.

[29] H. Wang. Games, Logic, and Computers. *Scientific American*, 98-106, November 1965.

[30] J. Weber and J. Penn. Creation and Rendering of Realistic Trees. In *Proceedings of SIGGRAPH 1995*, 119-128.

[31] L-Y. Wei and M. Levoy, Fast Texture Synthesis using Tree-structured Vector Quantization. In *Proceedings of SIGGRAPH 2001*, 479-488.

[32] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. Surface Splatting. In *Proceedings of SIGGRAPH 2001*, 371-378.